Appears in the Proceedings of SPIE 1995, Stereoscopic Displays and Virtual Reality Systems.

The *Bridge* between developers and virtual environments: A robust virtual environment system architecture

Rudy Darken[*]
Cynthia Tonnesen[*]
Kimberly Passarella Jones[‡]

[*]Naval Research Laboratory
Washington, D.C.
and
The George Washington University
Department of Electrical Engineering and Computer Science
Washington, D.C.
darken|tonnesen@enews.nrl.navy.mil

[‡]University of North Carolina at Chapel Hill
Department of Computer Science
Chapel Hill, N.C.
passarel@cs.unc.edu

ABSTRACT

A new approach is presented to the problems associated with designing and implementing virtual environment applications which supports robust methods of describing and building virtual worlds and interaction techniques. Despite the functional similarities between virtual world and graphical user interface (GUI) designers, virtual world designers are hindered by a lack of low level support which user interface designers typically get from a toolkit or application framework. Furthermore, a well-defined set of standard interaction techniques and devices has been established for the desktop metaphor which does not exist in the virtual environment arena.

The importance of shielding the programmer from the low level details of rendering, network communication, and device software is widely recognized. However, most off-the-shelf virtual reality systems provide only a device polling mechanism through which the entire interface must be built. Polling forces the programmer to delve into the low level details of each device for every interaction technique to be used. We are currently developing a system called the Bridge that is designed to meet the needs and requirements of highly interactive virtual environment applications. The distributed architecture of the Bridge clearly separates the interaction from the application and supports a more efficient, event-driven model. High level descriptions of user-computer dialogues can be easily constructed allowing modification of interaction techniques and styles.

Keywords: virtual environments, user-computer interfaces, dialogue, software architecture

1. INTRODUCTION

1.1. Motivation

As applications of virtual environments become more complex, so also must the interfaces to these applications. Consequently, virtual environment applications demand high performance computation and communication as well as support for robust interaction techniques and dialogue[†] structure. With each new approach to virtual environment system architecture design (of which there have been many), we find that overall performance is slowly increasing. Issues such as the animation frame rate, input response times, and multiprocess communication have been addressed. However, although the overall objective of these software architectures is to alleviate the implementation problems of the virtual world builder while preserving system performance, the assistance thus far has been shallow – particularly in the area of interface support.

Few systems offer any type of dialogue structure necessary for high level user interface construction. Thus the flexibility of the system to adapt to changing technology such as new hardware devices and interaction techniques is often

[†]. We describe a dialogue in this context as the enumeration of all user action(s)–system response(s) pairs. In other words, it is the language by which the user and the application “talk” to one another. We will elaborate on this point in section 3.

limited. Without dialogue support, each technique must be constructed in the application from low level device data which must be explicitly requested each time through the animation loop. This approach is nether reusable nor efficient, both in terms of system performance and programming effort.

This paper will expand on this notion of user interface dialogue as applied to virtual environments specifically focussing on areas we believe current technology falls short. A software architecture is described which is currently under development that addresses issues related to robust interface design and performance.

1.2. Requirements

In surveying the literature on software architectures for virtual environments, a number of relevant design requirements have been identified. Most of these have to do with maintenance of system performance. However, we have expanded on this and categorize design requirements in terms of performance issues and programming issues. Programming issues deal primarily in describing what the virtual world is while performance issues deal primarily with how the virtual world operates and executes the functionality as described. An in-depth discussion will follow in section 3.

Performance requirements

- Graphics frame rate. The frame rate must not lag below 10 frames per second to preserve the illusion of movement [1]. The consequence of a slow frame rate can be poor operator performance and often disorientation and simulator sickness.
- Response to user inputs. This requirement is flexible depending on the input in question. While it may be entirely acceptable to allow a small lag between the utterance of a spoken word and its acceptance, the same lag would be unacceptable for the movement and subsequent graphical update of a virtual hand device. Inputs which are closely associated with the frame rate (such as virtual hand data) must be updated at a comparable rate (≥ 10 updates per second). Response times of other user inputs must be determined within the context of the application.
- Communication speed. This is a generic requirement involving the time consumed in sending data between two points; either logically or physically. Under current computing power constraints, it is not possible to develop a functional virtual world beyond a simple toy world on a single processor. Thus efficient communication between multiple processors, multiple computers, and multiple software modules is necessary. This communication must consider not only hardware constraints but the bandwidth required by the application.
- Multiple simultaneous device support. Virtual environment interfaces are multimodal by their nature. Therefore, the use of multiple simultaneous input and output devices is required. This implies not only that several devices can be operated concurrently, but also that they can be combined to form higher level interactions. This will be further clarified in the discussion of dialogue.
- Logical device support. Logical devices (often called widgets [2]) are objects with geometry and associated behavior. They typically combine lower level interactions using a syntactical description into higher level events. Use of such objects is essential to the development of robust interfaces to virtual environments. In the same way that window environments have predefined widgets (menus, dialogue boxes, scroll bars, etc.), virtual environments can also benefit from the incorporation of reusable, familiar interaction techniques.
- Continuous and discrete event support. We separate events into two categories: continuous and discrete. A discrete event is an isolated occurrence of some importance. An object-object intersection or a spoken word are examples of discrete events. Continuous events, on the other hand, are not isolated but rather occur repeatedly for some duration of time. The movement of a tracker to control eye position is an example of a continuous event. Although both are important parts of a human-computer dialogue, they should be handled differently. Continuous events are closely associated with the frame rate while discrete events can be handled in a less timely manner.

Programming requirements

- Intuitive programming interface. Often overlooked in the design of a development environment is the organization of the system from the programmer's point of view. It is all too common to find toolkits designed from the perspective of the toolkit implementation rather than the end user (the programmer). For this reason, an object-oriented design is applicable with objects defined in terms of what the programmer needs to manipulate to accomplish the task at hand.
- Extensibility. Due to the dynamic nature of virtual environment technology in general, it is essential to allow for change in a virtual environment system architecture. As new device technology is introduced, network speeds increase, and computational power expands, the software must adapt without major alterations to the underlying fabric of the system.

- Flexibility. There are basically no universally accepted interaction devices or techniques in use today in the virtual environment arena. Therefore, a system design must account for the many possible combinations of device-technique-action mappings and support a rapid method of modification.
- Simulation process attachment. The importance of maintaining a high animation frame rate is widely accepted. The most common technique used to maintain frame rate is asynchronous simulation and animation processes as described by Robertson, et al. [3]. However, this is not a trivial implementation. Ideally, we would like to provide a mechanism by which heavy computation processes such as a fluid dynamics simulation can be executed on a separate process without sacrificing communication with the animation process which controls the interface. Furthermore, because this technique is so commonly used, it should be made as transparent as possible to the programmer.
- Dialogue description support. One of the primary topics of this paper is dialogue description and control. In order to support the development of high level interactions with an application, the developer must be able to build a “language” by which the operator will communicate with the system.
- Separation of content from style. A widely recognized issue in interface design in general is the need to separate the specifics of the application (the content) from the language of the interface (the style). In fact the User Interface Management System (UIMS) [4] was developed in part to deal with this issue. However, this separation can be taken too far resulting in an ineffective interface which is disjoint from the application [2]. The objective is to develop an interface which separates the content from the style in a modular fashion while maintaining integration with the application.

Each of these requirements will be discussed in more detail in section 3 where the Bridge architecture is described. All of these have been addressed in some form in previous work in this area. However, in reviewing the field, we find that there is a general difference of opinion as to what a virtual environment system architecture should be and consequently, no single system addresses all requirements. Often, one approach to a problem is chosen over another, although both may be valid due to the widely varying types of suitable applications for virtual environment technology. We will first review a number of approaches to virtual environment systems and what we view as the strengths and weaknesses of each. This will be followed by a description of the Bridge system architecture design.

2. BACKGROUND

The system most closely aligned with the Bridge design is the Veridical User Environment (VUE) developed at IBM Thomas J. Watson Research Center [5]. It stresses the importance of dialogue structure in virtual environment interfaces. However, we see two primary differences in our approach. First, the VUE system downplays the importance of interprocess communication stating that “a virtual world requires relatively low bandwidth among the individual processes...”. We dispute this claim on the grounds that many applications require extremely high bandwidth from simulation to interface. Were this not so, there would not be a need for specialized network protocols and systems such as Distributed Interactive Simulation (DIS) [6], NPSNET [7], and BrickNet [8]. A classic example of a high bandwidth application would be a simulation of a large number of autonomous objects. At each step in the simulation, the state of each of these objects (position, orientation, scale, color, etc.) may be updated. This introduces an extremely high network load demanding a more efficient method of communication. Second, although VUE contains the most thorough dialogue control and description mechanism to date, it seems to overlook the importance of polling in addition to, rather than instead of, event queueing. In other words, how does the programmer initiate the request of device (physical or logical) data? We believe that both polling and event queueing are necessary.

Appino, et al. [5] describe their dialogue structure in terms of three levels each being a subdialogue of the next higher level [9]: the specific, the generic, and the executive. At the lowest level is the specific level which refers to the hardware and software devices while the generic level deals with combining specific level events into interaction techniques. Lastly, the executive level handles event coordination and the attachment of semantic significance to user actions. This three level description is analogous to the lexical-syntactic-semantic designation described by Foley, et al. [10]. As is the case with graphical user interface (GUI) design, each level requires a separate design in order to determine what inputs and corresponding feedback constitute the interface. A goal of the Bridge design is to accommodate this three-level development.

The Minimal Reality (MR) toolkit developed at the University of Alberta [11] places a layer of abstraction in between the programmer and the necessary resources; graphics, devices, and computation. In taking this approach, MR does not handle event processing as a queueing mechanism but rather requires that all data requests be initiated by the application. For example, when the interface needs to know when a hand posture is made, it must specifically ask for that information rather than register an interest in this event and have it reported as it occurs. For discrete events, such as a hand posture, event queueing allows most processing to occur outside the animation loop and is therefore more efficient than polling. On a separate issue, while MR provides a programming interface to controlling processes and network resources, it forces the programmer to control multiple processes explicitly. This issue was addressed by the Distributed Virtual Environment Research Platform (DIVER) developed at the University of Virginia [12].

DIVER makes the control of asynchronous processes transparent. The programmer develops an application process which executes external to the renderer. The rendering process, in addition to handling the graphics, also controls input and output devices. Communication to/from the rendering process is accomplished via a network protocol. If the application needs to know the value of a tracker, a request is sent from the application to the rendering process which then returns a data packet with the requested information. Although the separation of the interface from the simulation is clearly defined and maintained in this system, it will have difficulties accommodating applications requiring a high bandwidth between the interface and the simulation due to the resulting high network traffic. Furthermore, similarly to MR, DIVER does not support dialogues at the same depth as VUE. Consequently, complex interaction techniques are more difficult to implement and are not necessarily reusable.

These systems each address specific issues and problems inherent to virtual environments. However, none of them cover all of the requirements described in section 1.2. The Bridge system architecture was designed to provide the low level support universally accepted as essential to virtual environment development (i.e. 3D hierarchical graphics, serial device control, communications support) while also addressing the issues of high level interface construction, specifically that of dialogue description and control.

3. THE BRIDGE SYSTEM ARCHITECTURE

3.1. Conceptual design

The overall structure of the Bridge is based on two primary principles. First, in order to maintain acceptable system performance, particularly the frame rate, the animation loop must compute and render only what is necessary for each frame eliminating all extraneous processing. Second, dialogue support is best facilitated by the separation of application content from interaction style. These two issues are closely coupled. As illustrated in figure 1, the application and/or the simulation are executed asynchronous to the rest of the system in each case. This increases system performance as well as separates the content of the simulation from the style of the presentation. However, our approach differs from others in that we do not confine content to an asynchronous process. Rather, as suggested by Conner, et al. [2], the Bridge supports a closer integration of content and style while still maintaining modular separation.

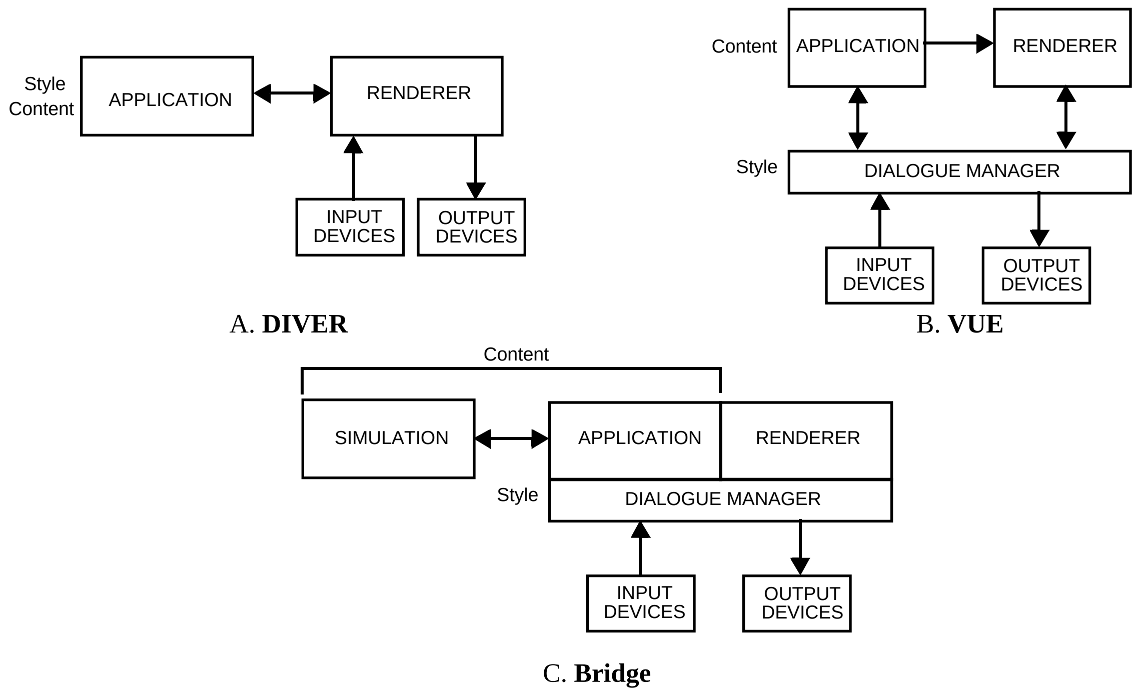

Figure 1: Schematic diagrams of the Bridge system architecture and its primary influential predecessors.

The distinction made between the terms *simulation* and *application* as used in figure 1C, has to do with level of content. We take simulation to mean that part of the system entirely devoted to content but which has no knowledge of the context of the application (i.e. it does not know it is being used in a virtual environment). The simulation is not the domain of the interface designer or the world builder. Recognizing that constructing a virtual environment entails much more than designing an interface, the term application is used to describe that part of the implementation which maintains content within the context of a virtual environment. It does not contain information dealing with style (i.e. the interface). As an example, consider a fluid dynamics system (See figure 2). The simulation computes the flow of the fluid

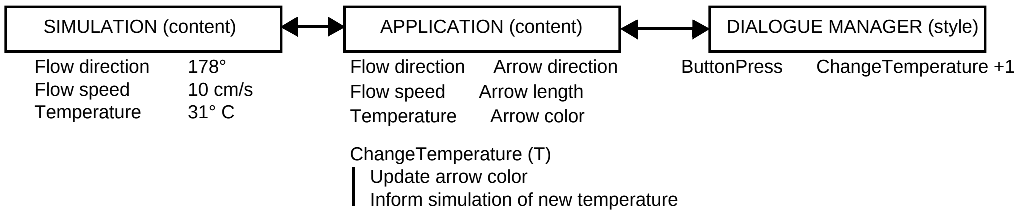

Figure 2: An example showing the conceptual distinction between the simulation, the application, and the dialogue manager.

producing data describing the environment (but not what it looks like or how to interact with it). The application connects the simulation to the renderer and the dialogue manager allowing the transfer of simulation data to the virtual environment and also of user requests back to the simulation. The application determines what the fluid flow looks like (i.e. how it is visualized). In this case, an arrow is used to encode flow direction, speed, and temperature. The application also provides the semantic descriptions of user requests. In the GUI domain, this is often a set of callback routines that are called to handle a particular input event. In this example, there is a description of what to do when the ChangeTemperature action is invoked. The description includes changing the color of the arrow and notifying the simulation of the change. The application has no information as to what user action signified a request to change the temperature. Only the dialogue manager knows that a button press performs this function. Subsequently, if the interface designer wishes to use a hand posture to control this function, only the dialogue manager needs to be changed. All aspects of interactive style are encapsulated in the dialogue manager.

By removing the computationally intensive simulation process(es) from the animation loop, the Bridge maintains a high graphics frame rate. This technique, however, introduces a communication problem. With the simulation now residing elsewhere, the rendering and simulation processes cannot interact as freely as they could as one process. While others have solved this problem using a network message passing scheme from process to process [5, 11, 12], we find that network limitations hinder the development of large scale applications. The Bridge addresses this issue using a shared memory mechanism allowing data to be shared between objects on multiple processes on a single machine. Both the simulation and the rendering processes have similar objects which share essential data. Communication between these processes requires time on the order of a memory access rather than that of an RPC call. From the programmer's point of view, the data resides in RAM and is always immediately accessible. This same mechanism is used to handle all time critical communication within the system.

At the heart of the Bridge is the dialogue manager which is the primary facility by which the programmer describes the interface. As shown in figure 1, style is encapsulated in the dialogue manager while the content is confined to the simulation which executes asynchronous to the rest of the system and to the application which, in part, holds the link between content and style. This approach allows the developer to minimize the "disposable" part of an application. That is, any part of the implementation which is specific to the application (i.e. it contains content knowledge) is not reusable. The Bridge encourages reusability by removing interaction technique descriptions from the application into the dialogue manager. This issue will be discussed in detail in the next section.

3.2. The Dialogue Manager

The dialogue manager of the Bridge can be viewed as an embedded transition network which maintains the "state" of the interface. At the lowest level (the lexical [10] or the specific [5] level), device details are encapsulated. Hardware capabilities are formed into atomic units. These units can be thought of as the "words" of the interface language. This includes items such as button presses, hand gestures, or any entity which cannot be subdivided further with any meaning. This level also is responsible for interacting with the device servers which, similar to the simulation process, execute asynchronous to the animation process where the dialogue manager resides. These atomic units (lexicons) are combined into "sentences" using syntactic rules (the syntactic [10] or the generic [5] level) which dictate how they can be legally composed. This is often where interaction techniques (also logical devices) are described. The third level (the semantic [10] or the executive [5] level) is where the "sentences" formed in the last level are understood. As an example, the popular "point and fly" style of virtual movement using a glove device and tracker would be described as the combination (syntax) of a finger pointing hand posture (a lexicon) and tracker orientation (a lexicon) into a higher level token (we will call it POINT). The dialogue then understands the POINT token to mean move the viewpoint in the direction specified by the tracker orientation. This same sequence of actions is executed in reverse for output events. This allows for the design of semantic, syntactic, and lexical feedback. Another benefit of this style of dialogue control is that it is flexible. Because the interface is described in a hierarchical fashion using subdialogues, it can be altered quickly at any level. For example, we can easily redefine viewpoint movement to be controlled by a button rather than a glove device without altering the viewpoint movement method.

This style of user action processing is an event queueing model. Rather than require the request and subsequent processing of input data explicitly in the animation loop, the dialogue manager is informed of what events are relevant and reports them accordingly to the application. Furthermore, since the dialogue manager is integrated with the rendering process, data associated with events can also be provided. If the interface requires the detection of intersecting objects, the dialogue manager can report not only the intersection but can also identify which objects collided. However, even though we rely on queueing for discrete events, low level data is available upon request (a polling mechanism) for continuous events. This is particularly useful in cases where the application requires access to device data. For example, when grasping a virtual object, the graphical representation of the object will move with the tracker it is attached to. Consequently, we would want tracker data continuously until the grasp is released. We have found that both polling and queueing are necessary in constructing efficient interfaces.

The fact that data producing devices operate at their own pace (usually faster than the frame rate) on separate processes allows fast responses to user inputs. There is no significant lag in response to data which is closely associated with the frame rate such as tracker data. The system easily adapts to new device technologies requiring only that a process be constructed to control the device and communicate with the dialogue manager.

4. CONCLUSIONS

One area we believe the Bridge can be improved upon is in developing a more complete separation of content from style. As it now exists, the dialogue manager encapsulates interaction style. However, visualization style is left in the domain of the application. We believe it is possible to provide a similar separation thereby confining visualization techniques to another conceptual module.

The Bridge is intended to provide virtual environment developers with the tools necessary to construct usable virtual worlds in which real work can take place. This implies that the system must not only support complex dialogue structures but also that it be flexible and efficient. The true test of this, or any, virtual environment system architecture will occur when it is applied to a computationally and conceptually complex simulation with an equally complex interface. Although computationally intensive applications [13] have been constructed, we have not yet seen an equally complex interface. Consider a compound interaction technique where dynamic gestures are combined with speech and application context with constraints. This forces the dialogue manager to deal with heavy computation simultaneously with communication with the application and the renderer. These types of scenarios will help to identify weaknesses in the Bridge's design and implementation and point the way to future improvements.

5. ACKNOWLEDGEMENTS

This research has been funded by the Effectiveness of Navy Electronic Warfare Systems (ENEWS) Program of the Tactical Electronic Warfare Division at the Naval Research Laboratory in Washington, D.C. The authors wish to thank Dr. Allen Duckworth and Dr. John Montgomery for their continued support of this work.

6. REFERENCES

[1] Card, S.K., T.P. Moran, and A. Newell, *The Psychology of Human-Computer Interaction*. 1983, Hillsdale, N.J.: Lawrence Erlbaum Associates.

[2] Conner, D.B., *et al. Three-Dimensional Widgets*. in *1992 Symposium on Interactive 3D Graphics*. 1992. Cambridge, Ma: ACM Press.

[3] Robertson, G.G., S.K. Card, and J.D. Mackinlay. *The Cognitive Coprocessor Architecture for Interactive User Interfaces*. in *User Interface Software and Technology '89*. 1989. ACM Press.

[4] Löwgren, J., *History, State and Future of User Interface Management Systems.* SIGCHI Bulletin, 1988. **20**(1): p. 32-44.

[5] Appino, P.A., *et al.*, *An Architecture for Virtual Worlds.* Presence: Teleoperators and Virtual Environments, 1992. **1**(1): p. 1-17.

[6] Institute for Simulation and Training, *The DIS Vision: A Map to the Future of Distributed Simulation.* 1994, Institute for Simulation and Training, Orlando, Fl.

[7] Zyda, M.J., *et al. NPSNET: Constructing a 3D Virtual World*. in *1992 Symposium on Interactive 3D Graphics*. 1992. Cambridge, Ma: ACM Press.

[8] Singh, G., *et al.*, *BrickNet: A Software Toolkit for Network-Based Virtual Worlds.* Presence: Teleoperators and Virtual Environments, 1994. **3**(1): p. 19-34.

[9] Jacob, R.J.K., *A Specification Language for Direct Manipulation User Interfaces.* ACM Transactions on Graphics, 1986. **5**(4): p. 283-317.

[10] Foley, J.D., *et al.*, *Computer Graphics: Principles and Practice*. 2nd ed. 1990, Reading, Ma: Addison-Wesley Publishing Company.

[11] Shaw, C., *et al. The Decoupled Simulation Model for Virtual Reality Systems*. in *SIGCHI '92*. 1992. Monterey, Ca: ACM Press.

[12] Gossweiler, R., *et al. DIVER: A Distributed Virtual Environment Research Platform*. in *IEEE 1993 Symposium on Research Frontiers in Virtual Reality*. 1993. San Jose, Ca: IEEE Computer Society Press.

[13] Bryson, S. and C. Levit, *The Virtual Windtunnel: An Environment for the Exploration of Three-Dimensional Unsteady Flows.* 1991, National Aeronautics and Space Administration, Ames Research Center